Pattern Recognition and Machine Learning Assignment

Ioannis Michalainas *10902*, Maria Charisi *10727*
Github Repo

December 2024/ January 2025

PART A

In this part, our study involves the quantification of levels of stress of video game players. Our task in is to implement a Maximum Likelihood Estimator to diversify between two classes ω_1 (stress) and ω_2 (no stress). We need to be able to accurately classify a player based on his play-style as either stressed or not stressed (binary classification).

Data

To perform our analysis we use the following data

- The Probability Density Function (PDF given θ)

$$p(x \mid \theta) = \frac{\pi}{1 + (x - \theta)^2}$$

- Training Sets D_1 and D_2 for both classes (ω_1, ω_2)
- A Discriminant Function

$$g(x) = \log P(x \mid \hat{\theta}_1) - \log P(x \mid \hat{\theta}_2) + \log P(\omega_1) - \log P(\omega_2)$$

A1

In this part we want to estimate variables θ_1 and θ_2. We begin by implementing the log likelihood function:

$$\log L(\theta \mid D) = \sum_{x \in D} \log p(x \mid \theta)$$

We now need to find θ_1 and θ_2 that maximize this function. A first approach would be to use the gradient of $logL(\theta \mid D)$ to find the optimal θ values. However, the gradient of this function does not have a closed-form expression, making it computationally challenging to apply this method directly.

$logL(\theta \mid D)$ maximization

Instead, we take a simpler approach:

- Define a range of candidate θ values that likely contains the true θ.
- Evaluate the log-likelihood function for these candidates and select the θ that maximizes it.

Since the data range spans approximately [-4.5, 4.1], we select a slightly wider candidate range for θ, such as [-5, 5], to ensure it includes the optimal value.

Plot

$\hat{\theta}_1$ estimation: 2.60 $\hat{\theta}_2$ estimation: -3.16

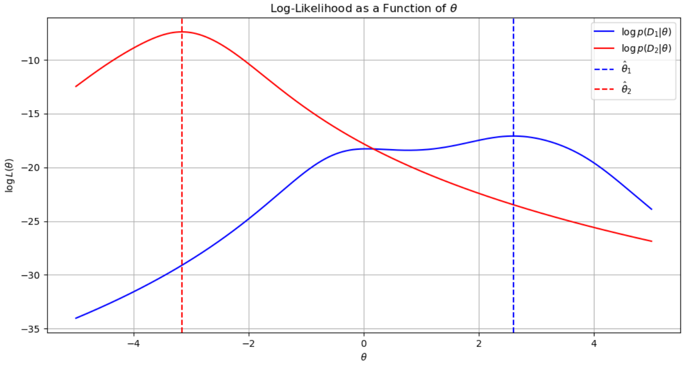


Figure: Visualizing Optimal θ Values

A2

Now, we need to use Discriminant Function

$$g(x) = \log P(x \mid \hat{\theta}_1) - \log P(x \mid \hat{\theta}_2) + \log P(\omega_1) - \log P(\omega_2)$$

to classify our data.

Priors Calculation

We calculate the *a priori probabilities* for each class ω_i. We have a total of 12 samples:

- 7 classified in ω_1
- 5 classified in ω_2

So we calculate $P(\omega_1) = 7/12$ and $P(\omega_2) = 5/12$

Plot

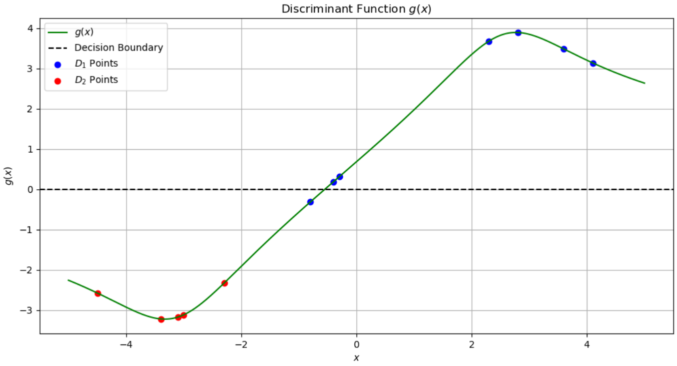


Figure: Discriminant Function Visualization

Observations

- Most of the $D1$ points are classified correctly, as $g(x) > 0$.
- All of the $D2$ points are classified correctly, as $g(x) < 0$.

Decision Rule: The decision boundary is at $g(x) = 0$. For any x:

- If $g(x) > 0$, classify x as 1 (no stress).
- If $g(x) < 0$, classify x as 2 (stress).

Conclusion

The classification rule leads to some misclassifications. While the decision rule works well for most of the data (11/12, 92%), there are always some trade-offs in classification accuracy. Achieving perfect classification is sometimes not feasible or desirable.

- Attempting to perfectly classify all points might lead to **overfitting**. A model that fits all training data perfectly may not generalize well to unseen data.
- The data may inherently contain some ambiguous or **overlapping** cases that no model can perfectly classify, especially if the two classes are not linearly separable.

Part B

In this part our task is to implement a new classifier for the previous problem, this time using Bayesian Estimation to estimate parameter θ.

Data

To perform our analysis we use the following data:

- The Probability Density Functions (PDF)

$$p(\theta) = \frac{1}{10\pi\left(1 + \left(\frac{\theta}{10}\right)^2\right)}, \quad p(x \mid \theta) = \frac{\pi}{1 + (x - \theta)^2}$$

- Training Samples D_1 and D_2 for both classes (ω_1, ω_2)
- A Discriminant Function

$$h(x) = \log P(x \mid D_1) - \log P(x \mid D_2) + \log P(\omega_1) - \log P(\omega_2)$$

B1

In this part, our goal is to calculate the a posteriori probability of θ,

$$p(\theta \mid D) = \frac{p(D \mid \theta) \cdot p(\theta)}{\int_{-\infty}^{\infty} p(D \mid \theta) \cdot p(\theta)\, d\theta}$$

θ Candidates

- In Part A, the dataset values were relatively small ([-4.5, 4.1]). Using a range of [-5, 5] was a practical choice because it encompassed the likely θ values where the likelihood peaks.
- In Part B, the prior distribution has a broader support (($\theta/10$) in the denominator suggests a wider possible range). To ensure the posterior distribution adequately integrates both the likelihood and the prior, we expanded the range to [-10, 10].

Plot

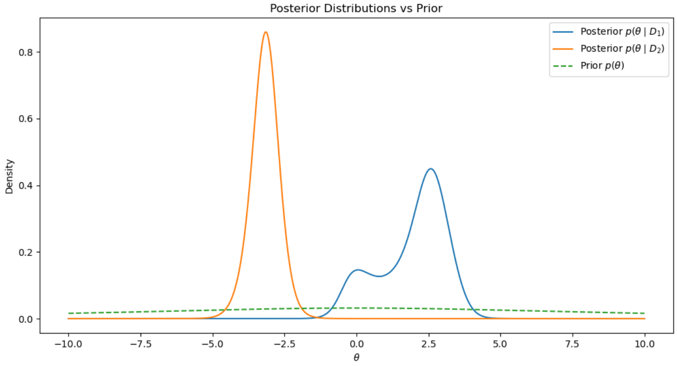


Figure: $P(\theta \mid D_1)$, $P(\theta \mid D_2)$ and $P(\theta)$

Observations

- The prior distribution $p(\theta)$ is flat and spread out over the range of θ values. It shows minimal preference for any specific θ, reflecting the prior belief before observing the data.
- The posteriors are much more concentrated than the prior, reflecting how the observed data updates the prior belief and provides more precise estimates of $p(\theta)$.
- The location of the peaks in the posteriors ($p(\theta \mid D_1)$ near 2 and $p(\theta \mid D_2)$ near -3) shows the influence of the datasets D_1 and D_2, respectively.

B2

We declare posterior predictive distribution $p(x \mid D)$ like so:

$$p(x \mid D) = \int p(x \mid \theta) p(\theta \mid D) d\theta$$

Then we declare the discriminant function $h(x)$:

$$h(x) = \log P(x \mid D_1) - \log P(x \mid D_2) + \log P(\omega_1) - \log P(\omega_2)$$

Plot

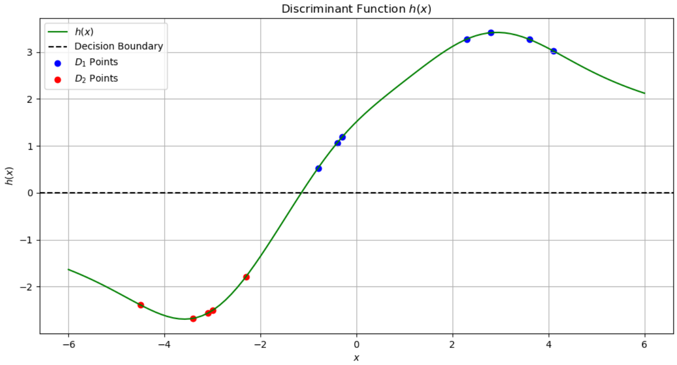


Figure: Discriminant Function Visualization

Observations

- All of the $D1$ points are classified correctly, as $h(x) > 0$.
- All of the $D2$ points are classified correctly, as $h(x) < 0$.

Decision Rule: The decision boundary is at $h(x) = 0$. For any x:

- If $h(x) > 0$, classify x as 1 (no stress).
- If $h(x) < 0$, classify x as 2 (stress).

Maximum Likelihood Estimation vs Bayesian Estimation

Part A:

- Estimates a single value for θ ($\hat{\theta}_1$ and $\hat{\theta}_2$) that maximizes the likelihood for each class.
- The decision boundary is determined by the log-likelihood and prior probabilities.

Part B:

- Considers the entire distribution of θ given the data (the posterior distribution) to make predictions.
- The decision boundary is determined by the posterior predictive distribution, which integrates over all possible values of θ weighted by their posterior probabilities.

We attribute the better performance of BE to the fact that we have prior knowledge about the parameter θ, in means of $p(\theta)$.

Part C

In this part, our study focuses on the classification of three Iris species: Iris setosa, Iris versicolor, and Iris virginica (i.e. three *classes*).

- Using the **sklearn** library, we analyze a dataset of 150 measurements (50 per species).
- Only the first two features (sepal length and width) are used.
- A DecisionTreeClassifier is implemented to classify 50% of randomly selected samples after training the model on the remaining 50%.

C1.1

Our goal is to find the optimal depth with respect to *accuracy*. To do so, we iterate in a range of possible depths (1,10), where the optimal depth should lie. We chose this particular range to avoid overfitting.

- Deep descision trees generally lead to overfitting -they tend to capture overly specific patterns that do not generalize well.
- This is especially true in our case, where the dataset is relatively small (150 samples).

Plot

We create the **Decision Tree** that yields the best *accuracy*. As shown in the plot, the optimal **depth** is 3 with **accurracy** 0.79.

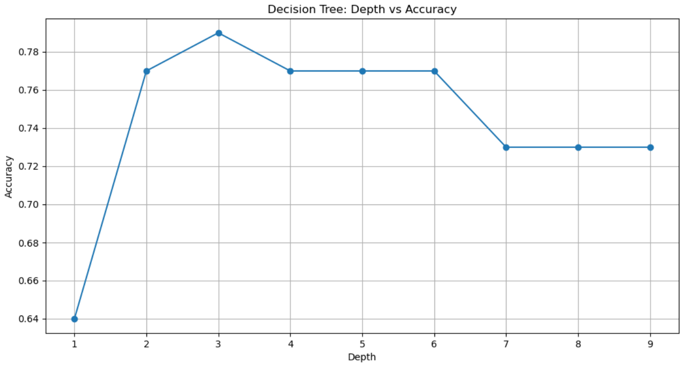


Figure: Optimal Depth

C1.2

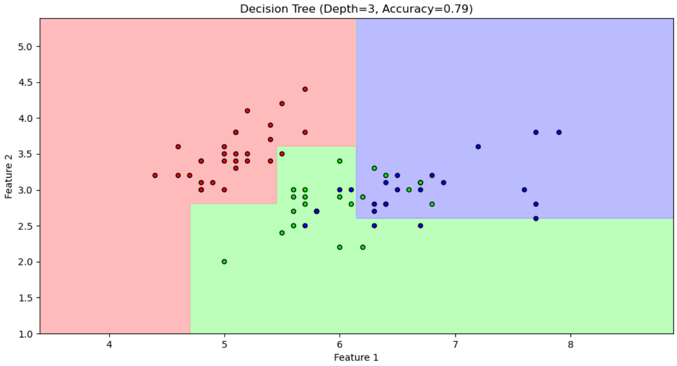


Figure: Classification

C2.1

In this task, we extend our classification of the three Iris species using a **Random Forest** classifier with *100 decision trees*.

- From the training set used previously (set A, 50% of the dataset), we create 100 training sets, each consisting of $\gamma = 50\%$ of set A, using the **bootstrap** method.
- Each decision tree is trained on its respective bootstrap set with a fixed maximum depth.
- The test set from the previous task is reused to evaluate the performance of the Random Forest classifier.

Optimal Depth

Our goal is yet again to find the optimal depth with respect to *accuracy*. To do so, we iterate in a range of possible depths (1,10), where the optimal depth should lie.

Plot

We create the **Random Forest** that yields the best *accuracy*. As shown in the plot, the optimal **depth** is 2 with **accurracy** 0.83.

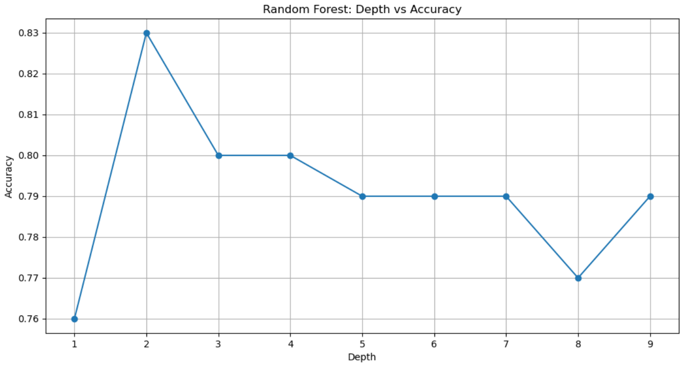


Figure: Optimal Depth

C2.2

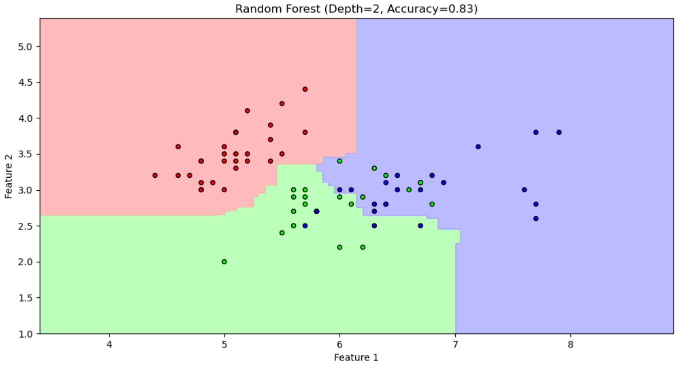


Figure: Enter Caption

C2.3

We calculate the *accuracy* achieved for different values of γ on its respective best depth. Essentially, we keep the *best* result we can get for each γ, as an example.

- For lower values of γ (0.1, 0.2, 0.3), the best accuracy varies between 0.80 and 0.81, with slight improvements as gamma increases.
- For higher values (0.4 and above), the accuracy stabilizes at around 0.83, indicating diminishing returns in performance improvement with increasing gamma.
- The best depth remains practically constant and equal to 2 for almost every γ.

γ Influence

- Generally, small γ leads to **higher bias**, because each bootstrap set contains less information about the entire dataset.
- On the other hand, for bigger values of γ the diversity among bootstrap sets decreases, which can reduce the ensemble's effectiveness at reducing variance.

In conclusion, increasing γ up to 0.4 seems to benefit our algorithm, but further increasing it does not have any effect on *accuracy* or the best depth for the Random Forest.

PART D

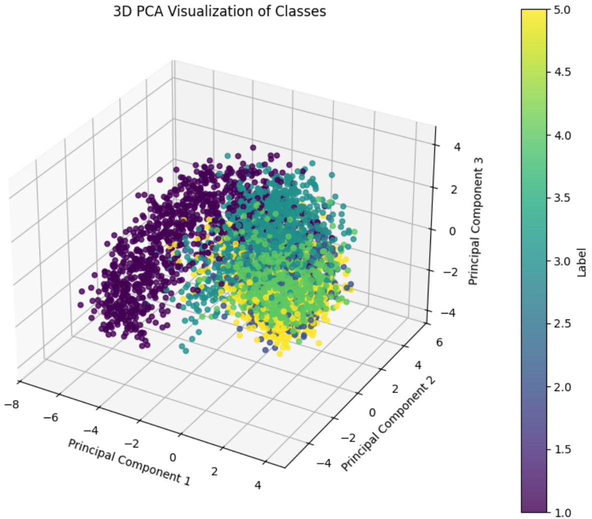
In this part, our task is to use datasetTV.csv to train a classifier of our choosing, which will be tested with datasetTest.csv. We are dealing with a classification problem with the following characteristics:

- 5-class classification problem
- 8743 training samples
- 224 features

Data Visualization

Our first move was to attempt to visualize our data. Since our data lie in high-dimensional space, it would be impossible to illustrate it. So, we performed PCA (dimensionality reduction) and projected the dataset in the 3D space, in an attempt to draw conclusions from the visual.

Data Visualization



Kernel Search

The visual representation is not of great help, but is a starting point for our analysis.
Our next attempt was to find a kernel in hopes that the dataset is **linear separable**. That would make approaches like:

- SVM
- Logistic Regression
- Naive Bayes

suitable for our classifier.

Silhouette Score

In order to check whether the transformed kernel data is linearly separable, we will use the **silhouette score** as a metric. The silhouette score quantifies how well a data point fits into its class and ranges from -1 to 1:

- 1: Perfectly matched to its class (linearly separable classes).
- 0: On the boundary between classes.
- -1: Misclassified, closer to another class (non-linearly separable classes).

Results

The silhouette score for the original dataset is 0.01.

Kernel	γ: **0.001**	γ: **0.01**	γ: **0.1**	γ: **1**	γ: **10**
rbf	0.03	0.03	-0.37	-0.03	-0.03
poly	0.03	-0.00	-0.07	-0.09	-0.09
sigmoid	0.03	0.03	0.03	0.03	0.04
cosine	0.04	0.04	0.04	0.04	0.04

Table: Silhouette Scores for Different Kernels and γ Values

Conclusions

As we can see from the above results, the silhouette scores of the original and transformed data range from -0.37 to 0.04. This indicates that our data are not linearly separable.

Testing Multiple Classifiers

To verify our hypothesis about the poor performance of linear approaches, we conducted a general search for the optimal classifier, using some default and generic parameters so that we get an idea of what performs best for our dataset.

Models

These are the models we tested with their respective parameters:

```
"Decision Tree": DecisionTreeClassifier(random_state=42),

"KNN(5NN)": KNeighborsClassifier(n_neighbors=5),
"Naive Bayes": GaussianNB(),

"Logistic Regression": LogisticRegression(max_iter=300, random_state=42),
"Perceptron": Perceptron(max_iter=300, random_state=42),
"Least Squares": RidgeClassifier(),

"SVM": SVC(kernel='linear', random_state=42),

"Neural Network": MLPClassifier(
    hidden_layer_sizes=(100,), max_iter=300, random_state=42),

"Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),

"AdaBoost": AdaBoostClassifier(n_estimators=100, random_state=42)
```

Cross-Validation Accuracy

To test the performance of every model we used **5-fold cross validation**:

- The dataset is randomly divided into 5 equal parts (or folds). Each fold is used once as a test set while, the other 4 are used for training.
- For each iteration, the model is trained on the training set and its accuracy is evaluated on the test set.
- Once all 5 iterations are complete, calculate the average of the accuracies from all folds to get the cross-validation accuracy.

5-Fold Cross-Validation

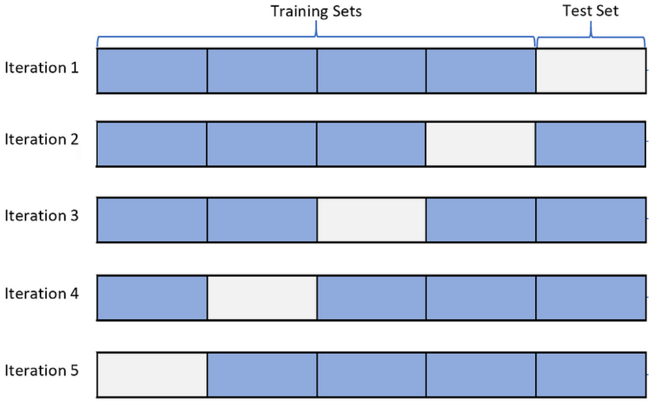


Figure: 5-fold Cross Validation

Results

Model	**Cross-Validation Accuracy**
Decision Tree	0.62
KNN(5NN)	0.81
Naive Bayes	0.70
Logistic Regression	0.77
Perceptron	0.69
Least Squares	0.75
SVM	0.76
Neural Network	0.83
Random Forest	0.80
AdaBoost	0.65

Table: Cross-Validation Accuracy for Different Models

Candidates

Out of all the classifiers we tested, we separated the

- k-Nearest Neighbors
- Random Forest
- Neural Network

classifiers as the best scoring in terms of accuracy.

These results align with our hypothesis that our data are not linear separable. Our goal now is to tune the parameters of said classifiers, in order to maximize the cross-validation accuracy and decide on our final model. To do so we performed a grid search for each one of them.

KNN Tuning

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski', 'cosine'],
    'p': [1, 2],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
```

Best cross-validation accuracy: **0.83**

Best parameters found:

- 'n_neighbors': 9
- 'weights': 'distance'
- 'metric': 'cosine'
- 'p': 1
- 'algorithm': 'auto'

RF Tuning

```
param_grid = {
    'n_estimators': [200, 300, 400, 500],
    'criterion': ['gini', 'entropy'],
    'max_depth': [10, 20, 50, None],
    'min_samples_split': [5, 10, 20],
    'min_samples_leaf': [2, 5, 10],
    'class_weight': ['balanced', None],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}
```

Best cross-validation accuracy: **0.83**
Best parameters found:

- 'n_estimators': 500
- 'criterion': 'entropy'
- 'max_depth': 20
- 'min_samples_split': 5
- 'min_samples_leaf': 2
- 'algorithm': 'auto'
- 'class_weight': 'balanced'
- 'max_features': 'sqrt'
- 'bootstrap': False

NN Tuning

```
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (200,), (100, 50), (100, 100)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd', 'lbfgs'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive'],
    'learning_rate_init': [0.001, 0.01, 0.0001],
    'early_stopping': [True, False],
    'max_iter': [200, 300, 500]
}
```

Best cross-validation accuracy: **0.84**
Best parameters found:

- 'hidden_layer_sizes': (200,)
- 'activation': 'relu'
- 'solver': 'adam'
- 'alpha': 0.0001
- 'learning_rate': 'constant'
- 'learning_rate_init': 0.01
- 'early_stopping': False
- 'max_iter': 200

Optimal Model

As we can observe, the model that achieves the best cross-validation accuracy for the training set is a Neural Network with the following characteristics:

- One hidden layer with 200 neurons
- ReLU activation function

$$\mathrm{ReLU}(x) = \max(0, x)$$

- Adam optimizer for weight updates
- Constant learning rate with an initial value of $\eta = 0.01$

Data Scaling

Although, before fitting the model to our data, we must preprocess it. First, we will **scale** the data, as this step is essential for models like neural networks.

Scaling improves the convergence of gradient descent and ensures that features with larger numerical ranges do not dominate the learning process.

Feature Selection

Moreover, our data lies in a high-dimensional space (224 dimensions), so we decided to use **feature selection**. Specifically, we used a correlation-based technique to discard features with low correlation to the label, as these are unlikely to provide meaningful information for the classification task. Using feature selection offers the following benefits:

- Reduce noise in the data
- Decreases the risk of **overfitting** because the model focuses on learning the underlying patterns rather than memorizing noise
- Improves computational efficiency

Aristotle University of Thessaloniki

FACULTY OF ENGINEERING
ARISTOTLE UNIVERSITY OF THESSALONIKI

Questions?

Pattern Recognition & Machine Learning
Assignment